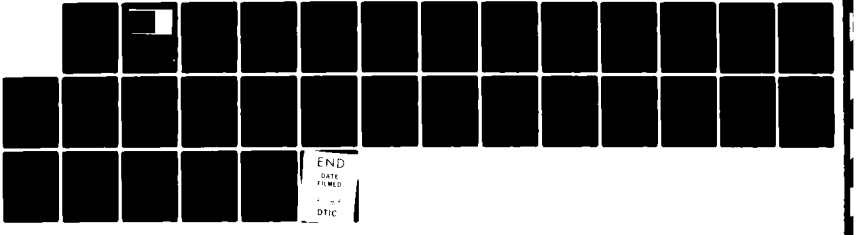
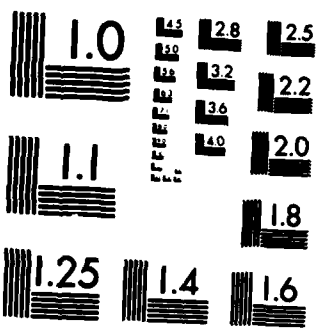
AD-A124 383 HIGHLIGHTS OF THE HISTORY OF THE LAMBDA-CALCULUS(U) 1/1
WISCONSIN UNIV-MADISON MATHEMATICS RESEARCH CENTER
J B ROSSER OCT 82 MRC-TSR-2441 DAAG29-80-C-0041

UNCLASSIFIED F/G 12/1 NL



END
DATE FILMED
DTIC



MRC Technical Summary Report #2441

HIGHLIGHTS OF THE HISTORY OF THE LAMBDA-CALCULUS

J. Barkley Rosser

Mathematics Research Center
University of Wisconsin—Madison
610 Walnut Street
Madison, Wisconsin 53706

October 1982

(Received October 12, 1982)

Sponsored by

U. S. Army Research Office
P. O. Box 12211
Research Triangle Park
North Carolina 27709

UNIVERSITY OF WISCONSIN-MADISON
MATHEMATICS RESEARCH CENTER

HIGHLIGHTS OF THE HISTORY OF THE LAMBDA-CALCULUS

J. Barkley Rosser

Technical Summary Report #2441
October 1982

ABSTRACT

This is an account of not only the lambda-calculus but of its close relative, the combinatory calculus. It begins with an introductory survey, so that no previous knowledge is required. It is explained why these are of such importance for computer software. The account is brought up to the present time. It includes the shortest and simplest proof of the Church-Rosser theorem, which is not yet published and appeared in a limited printing in August 1982. It includes a model of the combinatory calculus, also not yet published but available in 1982 in a limited printing. An introduction is given to some revolutionary new developments of the combinatory calculus for programming computers.

AMS (MOS) Subject Classifications: 01-A65, 03-03, 03-B40

Key Words: λ-calculus, combinatory calculus, foundations of programming

Work Unit Number 6 (Miscellaneous Topics)

Sponsored by the United States Army under Contract No. DAAG29-80-C-0041.

SIGNIFICANCE AND EXPLANATION

Programming for a computer went through a couple of major changes up to the invention of FORTAN. Since then, though many competing programming languages have been developed, the basics of programming have changed little. In the last half dozen years, some revolutionary new ideas for programming have appeared, involving the very fundamentals of the lambda-calculus and the combinatory calculus. By giving an account of these from the beginning, it is intended to make these revolutionary new ideas more easily comprehensible.

The responsibility for the wording and views expressed in this descriptive summary lies with MRC, and not with the author of this report.

HIGHLIGHTS OF THE HISTORY OF THE LAMBDA-CALCULUS

J. Barkley Rosser

Kleene-ness is next to Gödel-ness

1. EARLY BEGINNINGS.

The lambda-calculus originated in order to study functions more carefully. It was observed by Frege in 1893 (see van Heijenoort 1967, p. 355) that it suffices to restrict attention to functions of a single argument. For, suppose you wish a function to apply to A and B to produce their sum, A + B. Let Θ be a function, of a single argument, which when applied to A alone produces a new function, again of a single argument, whose value is A + B when applied to B alone. Note that Θ is not applied simultaneously to A and B, but successively to A and then B; application to A alone produces an intermediary function $\Theta(A)$, which gives A + B when later applied to B alone. That is, $A + B = (\Theta(A))(B)$.

This method of reducing the use of a function, "+", of two arguments to proper use of a related function, "Θ", of one argument only, is often referred to as "currying" because it was brought into prominence by the writings of H. B. Curry. Obviously, the method can be extended to reduce the use of a function of still more arguments to proper use of a related function of one argument only.

This is the way computers function. A program in a computer is a function of a single argument. People who have not considered the matter carefully may think, when they write a subroutine to add two numbers, that they have produced a program that is a function of two arguments. But what happens when the program begins to run, to produce the sum A + B? First A is brought from memory. Suppose that at that instant the computer is completely halted. What remains in the computer is a program, to be applied to any B that might be forthcoming, to produce the sum of the given A and the forthcoming B. It is a function

Sponsored by the United States Army under Contract No. DAAG29-80-C-0041.

of one argument, depending on the given A, to be applied to any B, to produce the sum A + B. It is Frege's intermediary function $\Phi(A)$.

Apparently Frege did not pursue the idea further. It was rediscovered independently (see Schönfinkel 1924) together with the astonishing conclusion that all functions having to do with the structure of functions can be built up out of only two basic functions, K and S. Let us adopt the notation that has been in vogue since then. Instead of writing the value that one gets by applying the function F to A as F(A), we write (FA). Omission of the outside parentheses will be usual. When more than two terms occur, association shall be to the left; thus MNP denotes ((MN)P), but M(NP) denotes (M(NP)). Then the sum of A and B would be written ΦAB.

The functions K and S are such that

(1.1) $$KAB = A$$

(1.2) $$SABC = AC(BC) .$$

For a proof that all functions can be built up of K and S, one can consult the original Schönfinkel paper. Or one can consult the two early papers, Curry 1929 or Curry 1930.

Expressions built up out of K and S by application (that is, enclosing pairs in parentheses) are called "combinators." Use of them, and study of their properties, is called "combinatory logic." Sometimes these terms are extended to apply when the expressions are allowed to contain variables, or indeterminates, as well as K and S.

Suppose we have two functions, F and G (built up out of K and S, of course) such that by means of (1.1) and (1.2) one can show that

(1.3) $$FX = GX$$

for each X, or for an indeterminate X. This means that F and G take the same value whenever they are applied to each X whatever, and so they ought to be the same function. That is, one should have

(1.4) $$F = G .$$

In general, one cannot prove (1.4) by means of (1.1) and (1.2). Curry (see Curry 1930) contrived additional axioms such that one can prove (1.4) whenever (1.3) holds for each X. A system like this is said to have the extensional property.

Curry added axioms to enable him to prove additional equalities. Did he go too far, so that now any two functions can be proved equal to each other? He did not. Indeed, he was careful to prove a weak form of consistency, in that there are many pairs of functions that cannot be proved equal to each other. And, especially, $K = S$ cannot be proved.

This gave a workable system, which illuminated many properties of functions. For instance, let M be built up from K, S, and the variable x. One can, using only K and S, build up a function F such that one can prove that

$$Fx = M$$

by means of (1.1) and (1.2). F turns out to be a mixed up combination of K's and S's. Just from looking at F, one would not have the least clue that $Fx = M$ should hold.

Because F is constructed in order to give the result $Fx = M$, it follows that

(1.5) $$FN = M[x:=N] ,$$

in which $M[x:=N]$ means the result of replacing each occurrence of x in M by N.

Church (see Church 1932) proposed that the F in question be called λxM. In this, M is intentionally part of the name of the function, so that by inspection you can see what you would get if you apply the function to x. For his construct, Church decreed that

(1.6) $$(\lambda xM)N = M[x:=N] ;$$

this accords exactly with (1.5).

If one starts with the left side of (1.6) and replaces it by the right side, this is called a β-reduction. One is equally entitled to start with the right side of (1.6) and replace it by the left side of (1.6); this is called a β-expansion.

Thus, to produce the Φ that we had earlier, Church would use $\lambda x(\lambda y(x+y))$. By (1.6), one would have

(1.7) $$(\lambda x(\lambda y(x+y)))A = \lambda y(A+y) .$$

By (1.6) again, one has

(1.8) $$(\lambda y(A+y))B = A+B .$$

So taking $\lambda x(\lambda y(x+y))$ to be $\oplus$, one has

(1.9) $$\oplus AB = (\lambda x(\lambda y(x+y)))AB = A+B$$

by two β-reductions.

The beauty of this is that at all stages of the process, one can tell by a simple inspection what the reduced form of $\oplus AB$ is going to be. This is the famous lambda-calculus of Church. (Henceforth, we will write LC for "lambda-calculus.") It does involve one with having to be careful about free and bound occurrences of variables. In $\lambda y(x+y)$, the *occurrence of* x *is free and both occurrences of* y *are bound.*

One has to be careful not to make manipulations which change free occurrences of a variable into bound ones. Thus, suppose one writes $\oplus y$. In this configuration, the observed occurrence of y is free. A blind adherence to (1.6) would give

$$\oplus y = \lambda y(y+y) ,$$

so that

$$\oplus yz = z+z .$$

This is certainly not what is intended for $\oplus$. The trouble is that the y, which originally existed as a free occurrence of a variable in $\oplus y$, has been put into $\lambda y(y+y)$ where its occurrence is now bound. Actually, when Church enunciated the rule (1.6) he was careful to impose the restriction that it should not be used if some variable with free occurrences in N should have those occurrences bound in $M[x:=N]$. (In addition, one must now understand $M[x:=N]$ to mean the result of replacing each <u>free</u> occurrence of x in M by N.) In order to cope with this contingency, Church instituted the α-step

(1.10) $$\lambda yM = \lambda z(M[y:=z]) .$$

NOTE: To avoid confusion of free and bound variables here, one must put the restrictions that there are no free occurrences of z in M, and that no occurrence of z in $M[y:=z]$ that resulted from replacing an occurrence of y in M by z is bound. Now we have

(1.11) $$\oplus y = (\lambda x(\lambda z(x+z)))y = \lambda z(y+z) ;$$

the intermediate formula is got from the left one by an α-step. So one has

(1.12) $$\oplus yx = (\lambda z(y+z))x = y+x \ ,$$

which is just what $\oplus$ is supposed to do.

If we can get from M to N by a succession of steps, possibly null, each of which is either an α-step or a β-reduction or a β-expansion, we say that M is convertible to N; we write "M conv N". If M conv N by a succession of steps, none of which is a β-expansion, we say that M is reducible to N; we write "M red N".

John McCarthy worked several ideas of the LC into LISP. He clearly recognized procedures as functions of one argument. In LC, such functions can be applied to each other and give such functions when applied. In LISP, it is possible to apply one procedure to another, and on occasion get another procedure.

As we said earlier, the K and S, and things built exclusively of them, are called combinators. Can we commingle combinators and lambda-expressions? Yes indeed, with no trouble whatever and indeed, variables, or indeterminates, may be freely included.

Church had decided that one should form λxM only in case there are free occurrences of x in M. Thus, Church could not get a lambda-expression to correspond to K. It could be done if one relaxes the requirement that there be at least one free occurrence of x in M to form λxM. So the LC, as originally set up by Church, seems a trifle weaker than the combinatory calculi of Schönfinkel and Curry. For present day applications, either would serve perfectly well (this takes some proving) and the difference is just something to niggle over, quite insignificant. Originally, this was not known, and Rosser (see Rosser 1935) invented a couple of other combinators, in place of K and S, with which he set up an exact equivalent of the LC. Like Curry, his system had the extensional property and a weak form of consistency. Hence the LC has these attributes also.

The LC (and hence, the combinatory calculi) has a fixed point theorem.

Given a function, F, one can find a ϕ such that

(1.13) $$F\phi = \phi \ .$$

Proof. Take

$$\phi = \psi\psi, \quad \text{where} \quad \psi = \lambda xF(xx) \ .$$

There is an obvious functional relationship between ϕ and F, namely

(1.14) $$\phi = YF ,$$

where

(1.15) $$Y = \lambda f((\lambda x f(xx))(\lambda x f(xx))) .$$

On pp. 177-179 of Curry and Feys 1958, Y is called the paradoxical combinator. The property

(1.16) $$F(YF) = YF$$

for each F is noted, which the authors thought to be paradoxical. The property (1.16) makes Y useful in some of the modern treatments of combinators. See p. 37 of Turner 1979 (first citation).

2. A DEBACLE.

The LC and the combinatory calculi were fairly promptly embedded in systems which had some of the earlier attributes of logical systems. See Church 1932 and Curry 1934. The results turned out to be inconsistent. This was first proved in Kleene and Rosser 1935 by a variation of the Richard paradox. Later, Curry got a simpler proof, related to the Russell paradox. See Curry 1942. This has the following simple form. Suppose we have the two familiar logical principles:

(2.1) $$P \supset P$$

(2.2) $$(P \supset (P \supset Q)) \supset (P \supset Q) ,$$

together with modus ponens (if P and $P \supset Q$, then Q). We undertake to prove an arbitrary proposition A. We construct a ϕ such that

(2.3) $$\phi = \phi \supset A ;$$

to do this, we take $F = \lambda x(x \supset A)$ in the fixed point theorem. By (2.1), we get

$$\phi \supset \phi .$$

Applying (2.3) to the second ϕ gives

$$\phi \supset (\phi \supset A) .$$

By (2.2) and modus ponens, we get

$$\phi \supset A .$$

By (2.3) reversed, we get

$$\Phi .$$

By modus ponens and the last two formulas, we get

$$A .$$

This is usually referred to as the Curry Paradox, by analogy with the Russell Paradox.

Fitch proposed to avoid this by weakening the LC (or equivalent combinatory calculus) so that the fixed point theorem fails. He also weakened modus ponens a bit. See Fitch 1936 and Fitch 1952. He has proved consistency for his system, but it is much too weak to be considered as a foundation for mathematics. He and his students have continued intermittently to the present to come out with improvements, but it still remains extremely weak.

Ackermann proposed keeping the full strength LC, but badly crippling implication. See Ackermann 1950 and Ackermann 1953. He proved the consistency of the system, but it was hopelessly weak as a foundation for mathematics, and I know of no recent interest in it.

Curry kept the full strength combinatory calculus, but added a fragmentary theory of types and a weakened version of implication. He introduces a notion of functionality, F, such that if Z is a function and X and Y are types, then FXYZ is to denote that if U is of type X, then ZU is of type Y. See Curry 1934 and Curry 1936. It turned out that the "natural" axioms for functionality lead to a contradiction, as shown in Curry 1955. However, by imposing suitable restrictions, all is well. See Curry 1956. Since then, Curry and his students have made extensive developments. Two major works, Curry and Feys 1958 and Curry, Hindley, and Seldin 1972, are landmarks. Even so, adoption of the system as a foundation for mathematics has not progressed at all, though the system has some capability in that direction; see Cogan 1955.

3. WHERE DO WE GO FROM HERE?

As we said, Fitch and Curry are continuing to develop their systems. However, there is no likelihood that either will be adopted as a foundation for mathematics. Originally, it was expected that the LC or a combinatory calculus should be a part of such a system. Surprisingly enough, the LC (or a combinatory calculus) has turned out to be of importance in its own right. So one has to ask the questions that are asked about any logical system.

1. What about consistency?
2. What about completeness?
3. What about models?
4. What about the connection with computers?

At the time when the LC and the combinatory calculi were being developed, one did not ask the fourth question. Computers had not yet been invented!

4. WHAT ABOUT CONSISTENCY?

We observed earlier that the LC and the combinatory calculi have a weak consistency, in that one cannot prove that both of any pair of functions are equal to each other. Fairly early on (see Church and Rosser '936; reproduced in Church 1941) a considerably stronger form of consistency was proved for the LC, embodied in the Church-Rosser Theorem (referred to hereafter as C-R-T).

Suppose X_0 red X_1 and X_0 red X_2. Then there is an X_3 such that both X_1 red X_3 and X_2 red X_3.

A lambda-formula is said to be in normal form if it has no part on which one can perform a β-reduction. A lambda-formula X is said to have a normal form Y if Y is in normal form and X conv Y. We can prove the following theorem.

If X has a normal form Y, then X red Y and Y is unique, except possibly for a few cosmetic α-steps.

One proves the first part of this by induction on the number of operations in X conv Y. The idea is as follows. Suppose one goes from X to W_1 by a β-expansion, next from W_1 to W_2 by a β-reduction, and finally from W_2 to Y by a

second β-reduction. Then W_1 red X and W_1 red Y. So, by C-R-T, there is a W such that X red W and Y red W. But Y is in normal form, so that in the reduction from Y to W there cannot be any β-reductions; only α-steps. So, except for cosmetic uses of α-steps, Y is W, and we had X red W. For uniqueness, suppose Z is another normal form of X. Then it is also a normal form of Y. So Y red Z. But Y is in normal form. So the reduction from Y to Z can consist only of α-steps.

The lambda-formulas of interest mostly have normal forms. These normal forms constitute a foundation, on which is erected an elaborate superstructure. *However, each* normal form has its own individual *superstructure, not* overlapping the superstructures of the other *normal* forms.

Because formulas of the LC can be identified with formulas of a combinatory calculus and vice versa, there are superstructures in the combinatory calculus corresponding to those of the LC.

In (1.1) and (1.2), one can consider going from left to right as a reduction. So one can look for parallels to C-R-T, one can define normal forms, etc. There was much investigation of these questions.

The original proof of C-R-T was fairly long, and very complicated. In Newman 1942, the point was made that the proof was basically topological. *Newman generalized* the universe of discourse, and defined a relation with properties similar to a β-reduction. He proved a result similar to C-R-T by topological arguments. Curry, in Curry 1952, generalized the Newman result, with the intention that it would be relevant to similar considerations in the combinatory calculi. Unfortunately, it turned out that neither the Newman result or the Curry generalization entailed C-R-T in the intended systems because the systems did not satisfy the hypotheses of the key theorems. This was discovered by David E. Schroer, whose counterexample is recorded in Rosser 1956. In Schroer 1965 is derived still further generalizations of the Newman and Curry results, which indeed do entail C-R-T in assorted systems. As Schroer 1965 is 627 typed pages, this hardly contributes to the cause of shorter and simpler proofs of C-R-T.

Chapter 4 of Curry and Feys 1958 is devoted to a proof of C-R-T for the LC, and to related matters. It is not recommended for light reading. In Hindley 1969 and Hindley 1974 are discussions of proofs of C-R-T for the LC and systems closely related thereto.

These various proofs all stemmed generally from the Newman approach, with an emphasis on the topological structure. However, lambda-formulas and combinators have a marked, though specialized, tree structure. Mitschke 1973 used the tree properties a bit in deriving a proof of C-R-T. Rosen, in Rosen 1973, really went overboard. He worked with general trees, and relationships between them. As lots of things have a tree structure, his results have applications beyond proving C-R-T. He applies his results to the extended McCarthy calculus for recursive definition (see McCarthy 1960), and verifies a conjecture in Morris 1968. He also applies his results to tree transducers in syntax-directed compiling. With all that, the proof of C-R-T did not come easy. He had to prove C-R-T's for several related systems, and then derive the C-R-T for the LC by some trickery.

Meanwhile, a genuine simplification for the proof of C-R-T had come in sight. See Martin-Löf 1972. It is agreed that Martin-Löf got some of his ideas from lectures by W. Tait. An exposition of the proof of C-R-T according to Tait and Martin-Löf appears as Appendix 1 in Hindley, Lercher, and Seldin 1972. A shorter exposition appears on pp. 59-62 of Barendregt 1981. We will give what seems to us a still shorter and more perspicuous proof of C-R-T.

What seems to be the main difficulty of the proof? Let us look at the minimal case. Suppose X_o has two parts, $(\lambda x W_1)V_1$ and $(\lambda x W_2)V_2$. Let X_o red X_i by performing a β-reduction on $(\lambda x W_i)V_i$, for $i = 1,2$. If $(\lambda x W_1)V_1$ and $(\lambda x W_2)V_2$ reside in totally disjoint parts of X_o, there is no trouble. To get X_3 we perform a β-reduction on the $(\lambda x W_2)V_2$ that still resides in X_1 and on the $(\lambda x W_1)V_1$ that still resides in X_2.

Note that the reductions from X_1 to X_3 and from X_2 to X_3 each use exactly one β-reduction.

But suppose that $(\lambda x W_2)V_2$ is part of V_1. X_2 will contain $(\lambda x W_1)V_3$, where V_3 is the result of a β-reduction of $(\lambda x W_2)V_2$ inside V_1. As a candidate for X_3, we perform a β-reduction on the $(\lambda x W_1)V_3$ of X_2. How about getting from X_1 to X_3?

Where X_o had $(\lambda x W_1)V_1$, X_1 will have $W_1[x:=V_1]$. If there had been only one free occurrence of x in W_1, then $W_1[x:=V_1]$ will contain a corresponding V_1; we change this to V_3 by a β-reduction on $(\lambda_2 W_2)V_2$,
contain several free occurrences of x. Then $W_1[x:=V_1]$ will contain several V_1's. We can go through, and change them one after another to V_3's, which will result in X_3. But there is no way we can get from X_1 to X_3 by a single β-reduction.

In the language of Barendregt 1981, p. 54, β-reduction does not have the diamond property.

The difficulty is that it may take several β-reductions to get from X_1 to X_3. This should have suggested working with a string of β-reductions, instead of only one. Why it took more than thirty years for this to occur to anyone is a mystery.

If we call a β-reduction or α-step a step, then a string of them will be a walk. But we cannot allow just any old string. What we are aiming for is that if X_o walk X_1 and X_o walk X_2, then there is an X_3 such that X_1 walk X_3 and X_2 walk X_3. If we put the right *restrictions on* the steps allowed in a walk, we can do this.

We frame our restrictions for a walk as follows.

1. A walk may contain no steps at all.

2. It may contain α-steps at will.

3. If a number of parts $(\lambda x W_i)V_i$ fail to overlap at all, the corresponding β-reductions may be done in any order.

4. Let $(\lambda x W)V$ be reduced to $W[x:=V]$ in a β-reduction of the walk. Inside that part, $W[x:=V]$, no subsequent β-reductions may be performed in the walk, and indeed no β-reduction of all of $W[x:=V]$, in case it has the requisite structure (which it could). However, α-steps may be performed inside $W[x:=V]$.

The relation $\xrightarrow[1]{}\!\!\!\!\rightarrow$ on p. 60 of Barendregt 1981 is likely closely related to our notion of a walk, but it is not exactly the same. For the key lemma, Barendregt uses something like induction on the number of steps from X_o to X_1 whereas we use induction on the number of symbols in X_o. This makes quite a difference.

We need a lemma, which is about as follows.

Suppose X walk Y. Then X[x:=P] walk Y[x:=P] by a completely parallel series of β-reductions.

To see this, note that α-steps do nothing to the free occurrences of x. A β-reduction can rearrange the free occurrences of x. It can even replicate them, as would happen if the β-reduction were from (λyS)T to S[y:=T]; if there are several free occurrences of y in S, they would each be replaced by T, and any free occurrences of x in T would be thereby replicated.

So, if occurrences of P are put for the free occurrences of x in X, a completely parallel series of β-reductions is possible, and all it will do is rearrange or replicate the P's just as the walk from X to Y did for the free occurrences of x. At the end, we just have Y[x:=P] as the result. The fact that the restrictions for a walk were satisfied in going from X to Y assures us that they will be satisfied in going from X[x:=P] to Y[x:=P].

Actually, the lemma is not quite true, because of the possibility of confusion of free and bound variables. Already, before you try the first step from X[x:=P] to Y[x:=P], you could be in trouble if some of the free variables in P became bound when P was put for x in X. However, a very close relative of the lemma, sufficient for our purposes, is true.

Lemma. Suppose X walk Y. In X, change all bound variables by α-steps to a set of distinct variables that have no occurrences in X or P. This gives X^1, for which there is a Y^1 such that X^1 walk Y^1 by essentially the same β-reductions as were used for X walk Y. Then X^1[x:=P] walk Y^1[x:=P] by a completely parallel series of β-reductions.

We first note that there will be no need for α-steps in either the walk from X^1 to Y^1 or from X^1[x:=P] to Y^1[x:=P]. All possibility of confusion of bound variables has been sidestepped in changing from X to X^1, and we can now use the argument given originally.

Note that this lemma is very nearly the same as proposition 2.1.17(i) on p. 28 of Barendregt 1981. It is also closely related to proposition 3.1.16 on p. 55. In Barendregt's terminology, our lemma says that a walk is substitutive.

Diamond Property. If X_o walk X_1 and X_o walk X_2, then there is an X_3 such that X_1 walk X_3 and X_2 walk X_3.

In other words, there is an X_3 which is the fourth vertex of the diamond, with a walk along each edge.

Proof by induction on the number of symbols in X_o.

Case 1. If X_o has a single symbol, it is immediate.

Case 2. Let X_o be $\lambda x M_o$. Then X_i must be $\lambda x M_i$ for i=1,2. Clearly we have M_o walk M_i for i = 1,2. So there is a M_3 such that M_i walk M_3 for i=1,2. Take X_3 to be $\lambda x M_3$.

Case 3. Let X_o be $M_o N_o$.

Subcase 1. X_i is $M_i N_i$ with M_o walk M_i and N_o walk N_i, all for i=1,2. Then there are M_3 and N_3 with M_i walk M_3 and N_i walk N_3, both for i=1,2. Take $X_3 = M_3 N_3$.

Subcase 2. M_o is $\lambda y W_o$, X_1 is $W_1[y:=N_1]$, and X_2 is $(\lambda y W_2)N_2$. By restriction 4, the last β-reduction in X_o walk X_1 had to be from $(\lambda y W_1)N_1$. So we have W_o walk W_i and N_o walk N_i, both for i=1,2. Then there are W_3 and N_3 such that W_i walk W_3 and N_i walk N_3, both for i=1,2. Then we have X_2 walk $(\lambda y W_3)N_3$, and hence we take X_3 to be $W_3[y:=N_3]$. There are various N_1's in $W_1[y:=N_1]$, but they are non-overlapping. So we operate on each in turn, and have X_1 walk $W_1[y:=N_3]$. By our lemma, $W_1[y:=N_3]$ walk $W_3[y:=N_3]$, the latter being X_3, except for some α-steps. Now the steps we took in going from $W_1[y:=N_3]$ were all on N_3's that had been put for y's in W_1. So none of them could violate restriction 4 as we go on down to $W_3[y:=N_3]$ from $W_1[y:=N_3]$. So we can put these two walks together to conclude X_1 walk X_3.

Subcase 3. Like subcase 2, except with X_1 and X_2 interchanged. Make suitable interchanges in the proof of subcase 2.

Subcase 4. M_o is $\lambda y W_o$, and X_i is $W_i[y:=N_i]$ for i=1,2. By restriction 4, the last β-reduction in X_o walk X_i had to be from $(\lambda y W_i)N_i$, both for i=1,2. So we have W_o walk W_i and N_o walk N_i, both for i=1,2. So there are W_3 and N_3 such that W_i walk W_3 and N_i walk N_3 both for i=1,2. There are various N_i's in $W_i[y:=N_i]$, but they are non-overlapping. So we operate on each in turn, and have X_i walk $W_i[y:=N_3]$, both for i=1,2. By our lemma, $W_i[y:=N_3]$ walk $W_3[y:=N_3]$, both for i=1,2, except for some α-steps. X_3 is $W_3[y:=N_3]$. To get from X_i down to X_3, we have to combine two walks, both for i=1,2, but the argument for this goes as in subcase 2.

Now we prove something that looks like C-R-T.

If X_o goes to X_i by a succession of walks, both for i=1,2, then there is an X_3 such that X_i goes to X_3 by a succession of walks, both for i=1,2.

The proof is so easy that, if we carry out the details for a special case, the whole thing becomes obvious. So, let X_o walk W_1 walk W_2 walk X_1 and X_o walk W_3 walk X_2. By the Diamond Property, we can fill in W_i's to be corners in Figure 1.

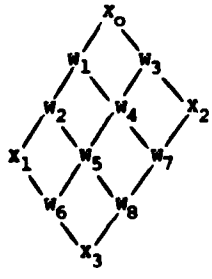


Figure 1

As each β-reduction or α-step taken alone is a walk, C-R-T follows by the previous result.

Although the proof in Newman 1942 failed to prove the C-R-T for the LC, it does prove a C-R-T for a fairly general universe of discourse. Some cases of this have been found to

be useful, though they have not much in common with the LC. See Book 1982, and several of the other authors cited in bibliographic references in Book 1982.

5. WHAT ABOUT COMPLETENESS?

At first sight, it appears that the LC is so weak that it is absurd even to raise the question. However, as indicated in Church 1932 and amplified in Kleene 1935 , the positive integers can be defined in the LC. If n is a positive integer, we let

$$\lambda f(\lambda x(f(f(\ldots(f(fx))\ldots)))) ,$$

where there are n f's, denote the integer n. This makes one form of recursive definition easy. If F(n) is to be defined by

(5.1) $F(1) = GA$

(5.2) $F(n+1) = G(Fn)$,

then we can take F to be

(5.3) $\lambda n(nGA)$.

With this definition,

F1 red GA

F2 red G(GA)

F3 red G(G(GA))

etc.

However, there is no zero in this system. One would prefer the recursive definition to be given by

(5.4) $F(1) = A$

(5.5) $F(n+1) = G(Fn)$.

In Kleene 1936, Kleene worked out a way to do this. This opened the door to still more general recursive definitions. More and more definitions of functions from integers to integers were discovered. Some never published investigations by Rosser disclosed so many that in about 1934 Church was led to conjecture that every effectively calculable function from positive integers to positive integers is definable in the LC. It was known from Church and Rosser 1936 that every function from positive integers to positive integers that

is definable in the LC is effectively calculable. So Church enunciated what is now known as "Church's Thesis."

Church's Thesis. Effectively calculable functions from positive integers to positive integers are just those definable in the LC.

As "effectively calculable" is an intuitive notion, Church's Thesis is *not* susceptible of proof. However, it states a strong, and quite unexpected, version of completeness.

In about this era, Gödel and Kleene were trying to get a definition for "general recursive function." Kleene gives a definition in Kleene 1936. He attributes it to Gödel. Gödel thought that general recursiveness should be taken as the criterion of effectively calculable. However, in Kleene 1936 it is shown that general recursiveness is the same as being definable in the LC. This lent strong support to Church's Thesis.

Independently, Turing had been developing the abstract idea of a computer, the so-called "Turing machine." See Turing 1936. Turing thought that "effectively calculable" should be taken to be the same as calculable on a Turing machine. But in Turing 1937, he proved that that is the same as being definable in the LC. This result explains why the lambda-calculus and the combinatory calculi can (and do) play such an important role in the theory of computer programming, and such matters.

Independently, in Post 1936, Post had developed ideas very similar to those of Turing. Turing published first, by a very few months. Later, in Post 1943, still another definition of "effectively calculable" was proposed, which turned out to be equivalent to those already given. Still later, in Markov 1951, Markov gave yet another definition, which was also proved to be equivalent. A translation of this appears as Markov 1961. In Smullyan 1961, using his "elementary formal systems," still another definition is given, which is also equivalent.

With the development of actual computers, which are finite approximations for a universal Turing machine, interest in all these matters has been much intensified. In Kleene and Vesley 1965, on p. 3, the authors list 150 contributions to the subject by October 15, 1963. By now there are far more.

Kleene's early developments in recursion theory were of much importance for computing. However, though he still uses many notations from the LC, he has diverged far from it into an area that is now essentially of no use in computing, though active and of interest to many people.

There have been some objections to Church's Thesis. In Moschovakis 1968 is given a simultaneous review of four papers, by Jean Porte, László Kalmár, Rózsa Péter, and Elliott Mendelson. The first three papers attempt in various ways to discredit Church's Thesis. The paper by Mendelson discusses the first three papers, and undertakes to show that their criticisms are illfounded. In the opinion of the reviewer, he succeeds quite adequately. I know of no recent attacks on Church's Thesis, and it seems to be generally accepted as an important, if unorthodox, version of completeness for the LC.

6. WHAT ABOUT MODELS?

There is a classic theorem that says that, if a logic is consistent, it will have a model; indeed a denumerable one. However, the LC is so different in structure from the usual logics that the theorem does not apply to it.

Why does one wish a model? If one has a framework with a lot of structure, and the logic is isomorphic to some part of the framework, then the structure in the framework can contribute to your understanding of the logic. One can always manufacture a very superficial model by taking equivalence classes of objects in the logic. The only structure this has is what is forced on it by the logic itself. So no additional understanding can come from studying the structure of the model. Such a model does little good.

For a very long time, this was the only kind of model that was found for the LC. Finally, with encouragement from Strachey, Dana Scott hit on a way of making some really useful models. They could be constructed either in the category of topological spaces or in the category of lattices. An exposition, "Outline of a mathematical theory of computation," appears in pp. 169-176 of the Proc. Fourth Annual Princeton Conf. on Information Sciences and Systems, 1970. In case this is inaccessible, another exposition

appears as the final article in Engeler 1971. In Barendregt 1981 is given a model similar to the Scott one, but in a still more general framework, namely the category of complete partial orders. In one sense, this is good since one can derive still more properties of the LC in this more general category. However, suppose one would like just to see a model without having to learn all the algebra involved in topological spaces, lattices, or complete partial orders. Some people have been working in that direction, to get a model without all the algebraic baggage. This is mostly available only in unpublished material, such as Plotkin 1972, Engeler 1979, and Meyer 1982; the latter gives a fairly complete and coherent account. According to Meyer, the model originated with Plotkin, was improved by Engeler, and further improved by Meyer himself. Our account is taken from the Meyer paper.

Start with a nonempty set, A; the unit class consisting of the ordered pair $\langle\phi,\phi\rangle$ will do, where ϕ is the null class. Enlarge A to the least set B containing A and all ordered pairs $\langle\beta,b\rangle$, where β is a finite subset of B and b is in B.

The model consists of all subsets of B. For two members, C and D, of the model, define

(6.1) $$(CD) = \{b\in B \mid \langle\beta,b\rangle\in C \text{ and } \beta\subseteq D\}.$$

To show that this contains a model of the combinatory calculus, we identify two elements K and S:

(6.2) $$K = \{\langle\alpha,\langle\beta,b\rangle\rangle \mid b\in\alpha \text{ and } \alpha,\beta \text{ finite subsets of } B\}.$$

(6.3) $$S = \{\langle\alpha,\langle\beta,\langle\gamma,b\rangle\rangle\rangle \mid b\in\alpha\gamma(\beta\gamma) \text{ and } \alpha,\beta,\gamma \text{ finite subsets of } B\}.$$

One verifies fairly easily that

(6.4) $$KCD = C$$

(6.5) $$SCDE = CE(DE)$$

for all elements of the model. A close relative of the extensional property holds; see Meyer 1982.

Since the *LC* is so closely related to the combinatory calculi, it is not surprising that something very similar can be put together as a model for the LC. In Meyer 1982 there are full details.

7. WHAT ABOUT THE CONNECTION WITH COMPUTERS?

This proceeds in two directions. One can use computers to manipulate combinators or formulas of the LC, or one can use properties of combinators and the LC to help in programming or to develop ideas of use for computers.

Looking to the first, the obvious approach would be to represent the combinators, or formulas of the LC, as lists or arrays in the computer memory. In fact, these formulas are tree structures, and might better be represented so on the computer. Knowing the location of only the root of the tree then suffices to reconstruct the entire tree. So the trees (entire formulas) can be identified by single memory locations, instead of by elaborate diagrams or linearizations thereof.

The idea is very simple. Suppose A and B are combinators, and we have put their roots at memory locations a and b. Then we represent C = (AB) by locating its root at memory location c; in c we put the ordered pair of numbers a and b. The person who wishes to know the structure of C is told to look at location c. There he finds <a,b>, which tells him that C has the form (AB), and that to know the form of A he should look in location a, and similarly for B.

Besides the convenience in referring to a formula, this allows economies of memory which are not possible when a formula is represented by a list. For an extreme example, suppose B = ((AA)(AA)), where A requires 1000 memory locations for its representation. To represent B as a list would require four repetitions of the listing of A, together with attendant parentheses; a total of 4006 locations. With the tree representation, let A have its root at a; we may still suppose that the entire representation of A fills 1000 locations. At some convenient memory location, d, we put <a,a>, which denotes D = (AA). Then at another empty memory location, b, we put <d,d>, which denotes (DD). But (DD) is ((AA)(AA)) = B. Thus, with A represented in 1000 memory locations, we require only 1002 locations to represent ((AA)(AA)).

Aother advantage of the tree representation is that it lends itself to what is called "lazy evaluation." Suppose a part M occurs several times in a formula X. If X is represented as a list, the several occurrences of M are each written out in full. Unless

extraordinary measures are taken, each of the occurrences of M will be evaluated separately, and independently, in the course of evaluating X. However, with a tree structure, M will occur only once, but with various pointers "pointing" to it. Hence, it will be evaluated once only.

These, and many related matters, are taken up in Petznick 1970. Consider a typical program on a computer, say for computing an approximation to the square root (two integers, a mantissa and an exponent). If one inputs an approximation for a real number (a mantissa and an exponent) the program will generate and output an approximation for the square root. So the program defines a function. Naturally, it is a computable function. So (by one of the equivalences supporting Church's Thesis) this function must be expressible by means of a combinatory formula. If suitable hardware, or software simulations thereof, is available, the calculation can be done solely by combinatory manipulations.

Something of the sort had been proposed for lambda-formulas by Landin. For this purpose, he defined and used what he calls SECD machines. See Landin 1965 or "A formal description of ALGOL 60," pp. 266-294 in Steel 1966. However, this involved him in a very difficult problem of handling the complicated substitutions properly. If he had used combinatory formulas instead, this problem would be much simplified. Also, Landin tried to superpose the lambda-formulas on top of the usual computer software. This produced a greatly complicated assignment problem. If one would dispense with the usual computer software, and work only with combinatory formulas stored in the memory (preferably as trees) the assignment problem would simply disappear.

Petznick's thesis, Petznick 1970, showed that it is possible to design a computer to work exclusively with combinatory formulas, stored as trees. There is no assignment problem, and application takes the place of substitution. As application is the basis of the tree structure, it is handled automatically. The hardware one would have to build to handle this would be quite simple. Or it can be handled with present hardware by a suitable software simulation.

Petznick's thesis managed to evade everybody's attention, and nothing more was done in that area for a while. But after some years, work similar to Petznick's, and extending it,

began to appear, and has quickly blossomed. It now engages the attention of a considerable number of people, all of whom seem to be quite unaware of Petznick's work.

There is quite a ferment of activity just now, and several papers were presented at the 1982 ACM Symposium on LISP and Functional Programming at Pittsburgh; a set of Proceedings is available under ACM order number 552820. It would surpass my powers as a soothsayer to determine what will emerge as the key ideas; perhaps some have not yet emerged.

I will sketch a couple of trains of development, to give the reader some sort of idea what is happening. In so doing, I may fail to note something that will be of major importance, and so fail to give credit due to those who are working on it.

In Henderson and Morris 1976 appeared an idea for lazy evaluation. The two papers, both cited as Turner 1979, carried this forward, and also showed how to condense combinatory formulas very much, thereby alleviating what had been a problem for Petznick. More on that last point is given in Hughes (to appear). There are now programs for manipulating combinators directly. One is given in CRS/1. Another is SKIM, which was announced in 1980, and is now being improved by a group at Cambridge University. Backus 1978 does not seem to be in the main stream of this activity, but it has some quite novel combinatory functions, and something interesting may evolve out of it.

It seems to be now established that operating directly on computers in combinatory format is not only feasible, but has some advantages. Even more useful results may be just around the corner. Or they may have already been announced without my appreciating their worth.

BIBLIOGRAPHIC INFORMATION

Most references below are cited by author and date alone, as in "van Heijenoort 1967." Some are without author, as "CRS/1." Some references below are to unpublished theses. Copies of some of these can be obtained from

University Microfilms International

A Xerox Publishing Company

300 N. Zeeb Road

Ann Arbor, MI 48106

Ackermann, W., "Widerspruchsfreier Aufbau der Logik I. Typenfreies System ohne Tertium non datur," Jour. Symb. Logic, vol. 15 (1950), pp. 33-57.

Ackermann, W., "Widerspruchfreier Aufbau einer typenfreier Logik. I," Math. Zeit., vol. 55 (1952), pp. 364-384; "II," Math. Zeit., vol. 57 (1953), pp. 155-166.

Backus, John, "Can programming be liberated from the von Neumann style? A functional style and its algebra of programs," Comm. Assoc. Comp. Mach., vol. 21 (1978), pp. 613-641.

Barendregt, H.P., "The lambda calculus," North-Holland Publ. Co., 1981.

Book, Ronald V., "Confluent and other types of Thue systems," Jour. Assoc. Comp. Mach., vol. 29 (1982), pp. 171-182.

Church, Alonzo, "A set of postulates for the foundation of logic," Annals of Math., second series, vol. 33 (1932), pp. 346-366.

Church, Alonzo, "The calculi of lambda-conversion," Annals of Math. Studies 6, Princeton Univ. Press, 1941; 2nd ed., 1951.

Church, Alonzo, and Rosser, J. B., "Some properties of conversion," Trans. Amer. Math. Soc., vol. 39 (1936), pp. 472-482.

Cogan, Edward J., "A formalization of the theory of sets from the point of view of combinatory logic," Zeit. Math. Logik Grundlagen Math., vol. 1 (1955), pp. 198-240.

CRS/1 specification, available from Beale Electronic Systems Ltd., Wraysbury, UK.

Curry, H. B., "An analysis of logical substitution," Amer. Jour. Math., vol. 51 (1929), pp. 363-384.

Curry, H. B., "Grundlagen der kombinatorischen Logik," Amer. Jour. Math., vol. 52 (1930), pp. 509-536.

Curry,, H. B., "Some properties of equality and implication in combinatory logic," Annals of Math., second series, vol. 35 (1934), pp. 849-860.

Curry, H. B., "Functionality in combinatory logic," Proc. Nat. Acad. Sci. U.S.A., vol. 20 (1934), pp. 584-590.

Curry, H. B., "First properties of functionality in combinatory logic," Tôhoku Math. Jour., vol. 41 (1936), pp. 371-401.

Curry, H. B., "The inconsistency of certain formal logics," Jour. Symb. Logic, vol. 7 (1942), pp. 115-117.

Curry, H. B., "A new proof of the Church-Rosser theorem," Nederl. Akad. Wetensch. Ser. A, vol. 55 (1952), (Indag. Math., vol. 14), pp. 16-23.

Curry, H. B., "The inconsistency of the full theory of combinatory functionality (Abstract)," Jour. Symb. Logic, vol. 20 (1955), p. 91.

Curry, H. B., "Consistency of the theory of functionality (Abstract)," Jour. Symb. Logic, vol. 21 (1956), p. 110.

Curry, Haskell B., and Feys, Robert, "Combinatory logic," North-Holland Publ. Co., 1958.

Curry, Haskell B., Hindley, J. Roger, and Seldin, Jonathan P., "Combinatory logic, vol. II," North-Holland Publ. Co., 1972.

Engeler, E., editor, "Symposium on semantics of algorithmic languages," Lecture Notes in Mathematics, No. 188, Springer-Verlag, 1971.

Engeler, E., "Algebras and combinators," Berichte des Inst. fur Informatik, Nr. 32, ETH Zurich, 12pp., 1979.

Fitch, F. B., "A system of formal logic without an analogue to the Curry W operator," Jour. Symb. Logic, vol. 1 (1936), pp. 92-100.

Fitch, F. B., "Symbolic logic, an introduction," The Ronald Press Co., New York, 1952.

Henderson, P., and Morris, J. H., "A lazy evaluator," Proc. 3rd annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, pp. 95-103, Atlanta, 1976.

Hindley, R., "An abstract form of the Church-Rosser Theorem. I," Jour. of Symbolic Logic, vol. 14 (1969), pp. 545-560.

Hindley, R., "An abstract Church-Rosser Theorem. II: Applications," Jour. of Symbolic Logic, vol. 19 (1974), pp. 1-21.

Hindley, J.R., Lercher, B., and Seldin, J.P., "Introduction to combinatory logic," London Mathematical Society Lecture Note Series 7, Cambridge Univ. Press, 1972.

Hughes, R. J. M., "Graph-reduction with supercombinators," Oxford University Programming Research Group technical monograph PRG-28 (to appear).

Kleene, S. C., "A theory of positive integers in formal logic," Amer. Jour. Math., vol. 57 (1935), pp. 153-173, pp. 219-244.

Kleene, S. C., "λ-definability and recursiveness," Duke Math. Jour., vol. 2 (1936), pp. 340-353.

Kleene, S. C., and Rosser, J. B., "The inconsistency of certain formal logics," Annals of Math., second series, vol. 36 (1935), pp. 630-636.

Kleene, S.C., and Vesley, R.E., "The foundations of intuitionistic mathematics," North-Holland Publ. Co., 1965.

Landin, P.J., "A correspondence between ALGOL 60 and Church's lambda-notation," Comm. Assoc. Computing Machinery, vol. 8 (1965), Part I, pp. 89-101, Part II, pp. 158-165.

Markov, A. A., "Teoriya algorifmov (Theory of algorithms)," Trudy Mat. Inst. Steklov, vol. 38 (1951, pp. 176-189.

Markov, A. A., "Theory of algorithms," No. OTS 60-51085, U.S. Department of Commerce, Office of Technical Services 1961.

Martin-Löf, P., "An intuitionistic theory of types," manuscript, University of Stockholm, 1972.

McCarthy, J., "Recursive functions of symbolic expressions and their computation by machine," Comm. Assoc. Computing Machinery, vol. 3 (1960), pp. 184-195.

Meyer, Albert R., "What is a model of the lambda calculus?" 1982. To appear in Information and Control, vol. 52.

Mitschke, G., "Ein algebraischer Beweis für das Church-Rosser Theorem," Archiv. für mathematische Logik, vol. 15 (1973), pp. 146-157.

Morris, J.H., Jr., "Lambda-calculus models of programming languages," MAC-TR-57, M.I.T. Project MAC, Cambridge, Mass. 1968.

Moschovakis, Y. N., A review, Jour. Symb. Logic, vol. 33 (1968), pp. 471-472.

Newman, M. H. A., "On theories with a combinatorial definition of "equivalence,"" Annals of Math., second edition, vol. 43 (1942), pp. 223-243

Petznick, George W., "Combinatory programming," Ph.D. thesis, Univ. of Wisconsin, Madison, Wis., 1970. Publication 70-24812 of University Microfilms Int.

Plotkin, G.D., "A set-theoretical definition of application," Memorandum MIP-R-95, School of Artificial Intelligence, Univ. of Edinburgh, 32 pp., 1972.

Post, Emil L., "Finite combinatory processes. Formulation I," Jour. Symbolic Logic, vol. 1 (1936), pp. 103-105.

Post, E. L., "Formal reductions of the general combinatorial decision problem," Amer. Jour. Math., vol. 65 (1943), pp. 197-215.

Rosen, B.K., "Tree manipulation systems and Church-Rosser theorems," Jour. Assoc. Computing Machinary, vol. 20 (1973), pp. 160-187.

Rosser, J. B., "A mathematical logic without variables," Annals of Math., second series, vol. 36 (1935), pp. 127-150, and Duke Math. Jour., vol. 1 (1935), pp. 328-355.

Rosser, J. Barkley, Review of Curry, "A new proof of the Church-Rosser theorem," Jour. of Symbolic Logic, vol. 21 (1956), p. 377.

Schönfinkel, Moses, "Über die Bausteine der mathematischen Logik," Math. Ann., vol. 92 (1924), pp. 305-316.

Schroer, David E., "The Church-Rosser theorem," Ph.D. thesis, Cornell Univ., June 1965. Publication 66-41 of University Microfilms, Int.

Smullyan, R. M., "Theory of formal systems," Annals of Math. Studies 47, Princeton Univ. Press, 1961.

Steel, J.B., Jr., editor, "Formal language description languages for computer programming," Proc. IFIP Working Conference on Formal Language Description Languages, North-Holland Publishing Co., 1966.

Turing, A. M., "On computable numbers, with an application to the Entscheidungsproblem," Proc. London Math. Soc., second series, vol. 42 (1936), pp. 230-265. "A correction," vol. 43 (1937), pp. 544-546.

Turing, A.M., "Computability and λ-definability," Jour. of Symbolic Logic, vol. 2 (1937), pp. 153-163.

Turner, D. A., "A new implementation technique for applicative languages," Software-Practice and Experience, vol. 9 (1979), pp. 31-49.

Turner, D. A., "Another algorithm for bracket abstraction," Jour. Symb. Logic, vol. 44 (1979), pp. 267-270.

van Heijenoort, Jean, "From Frege to Gödel," Harvard Univ. Press, Cambridge, 1967.

JBR/ed

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 2441	2. GOVT ACCESSION NO. AD-A124 383	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Highlights of the History of the Lambda-Calculus		5. TYPE OF REPORT & PERIOD COVERED Summary Report - no specific reporting period 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) J. Barkley Rosser		8. CONTRACT OR GRANT NUMBER(s) DAAG29-80-C-0041
9. PERFORMING ORGANIZATION NAME AND ADDRESS Mathematics Research Center, University of Wisconsin 610 Walnut Street Madison, Wisconsin 53706		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Work Unit Number 6 - Miscellaneous Topics
11. CONTROLLING OFFICE NAME AND ADDRESS U. S. Army Research Office P.O. Box 12211 Research Triangle Park, North Carolina 27709		12. REPORT DATE October 1982 13. NUMBER OF PAGES 26
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

λ-calculus, combinatory calculus, foundations of programming

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This is an account of not only the lambda-calculus but of its close relative, the combinatory calculus. It begins with an introductory survey, so that no previous knowledge is required. It is explained why these are of such importance for computer software. The account is brought up to the present time. It includes the shortest and simplest proof of the Church-Rosser

(continued)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT (continued)

theorem, which is not yet published and appeared in a limited printing in August 1982. It includes a model of the combinatory calculus, also not yet published but available in 1982 in a limited printing. An introduction is given to some revolutionary new developments of the combinatory calculus for programming computers.